# 加餐｜高并发场景：如何提升对突发事件的应急处理能力？

2021-10-11 胡丽麟

《手把手带你搭建秒杀系统》 课程介绍 >

**讲述：张浩**

时长 15:01 大小 13.76M

你好，我是胡丽麟，新浪微博的高级架构师，很开心受邀来到志东老师的专栏做一期加餐分享，今天我们来聊一聊高可用高并发服务。

最开始选择话题的时候，我其实有点迟疑，高可用高并发服务应该当属互联网技术分享的头把交椅了，各家技术争相斗艳层出不穷，包括我们这门课也会有相关设计思路的讲解，所以一番思想斗争之后，我决定和你谈谈自己一路过来的感想，包括我的成长经历以及一些阶段性的思考和沉淀，相信能给你带来启发。

说起高可用、高并发，我们很容易就联想起分布式、云服务、弹性扩缩容、微服务、纵架构、业务解耦、异步队列等等，一系列的技术和规范，优秀的架构师往往能设计出一套容易部署、容易扩展、可维护性较好的架构。

在微博发展的几个阶段里，架构也进行过多次升级，都是为了更好地应对高并发流量，提高服务的稳定性。但即便如此，突发热点还是会让整个系统及技术人员措手不及，那么为什么宣称能支持 N 个并发热点事件的系统在面对突增流量的时候还是会出现不可用情况呢？

有很多原因，比如通用架构面临某个单一事件的集中爆发：明星单个人的热点引发资源单点问题；再比如千里之堤上被人忽视的蚁穴：一个已经出问题的服务在热点事件中扩大了影响倍数。这也都是诸多互联网产品所会面临的一些挑战，想要攻克，解决方案一定是多方考量的，比较复杂。所以这节课我们不妨本着收获最大化的原则，从技术人个人层面出发聊聊**如何应对高并发场景中的突发事件**，我想这也是更具普适性的收获。

## 高并发的挑战

我刚加入微博的时候，恰逢微博快速发展初期，我们时常会遇到各种各样的问题，流量高了、服务不可用了，往往这个时候办公区里就特别沸腾，指点江山、排兵布阵、奋勇杀敌。

当时作为新生的我很是羡慕，一直觉得很威风。可时间长一点我就发现，这些光环总是在小部分人身上，尽管我们都在同一套高可用高并发的架构体系中进行几乎同样的业务需求开发，但遇到突发状况时能站在前面的都是那一部分人。此后我就开始思考**为什么他们能站在前面，我什么时候也能站到前面？**

所以我就开始观察、取经，慢慢地就发现这些人其实存在共性，他们都拥有良好的应急处理能力，而具备这种能力的人一般都具备这些素质：

**过硬的心理素质。**首先你得能做到不自乱阵脚，从容面对。

**良好的分析问题、解决问题的能力。**除了对本身系统有足够的了解外，他们一般对整个服务的链路、依赖关系、底层原理也都有一定的了解，能敏锐地捕捉到异常信息，抽丝剥茧找出问题根因。

**决策能力。**能对突发问题快速作出决策，降低故障时长、减少影响范围。

## 提升应急处理能力

当然了，这些描述更侧重表象，不妨再往深层想一想。评估架构设计能力，我通常会看他们的设计思路和实现技巧，看他们是怎么把复杂的问题梳理清楚，并按照逻辑关系组织编

码的。

总结来看，要想从容应对突发问题，并在更短的时间内做出判断、决策和执行，提高个人的应急处理能力，我认为还得从**细节**和**习惯**做起。

1. **打破自己的视野局限**

在一些复杂庞大的项目中，同时参与的部门、人数很多，大家协同完成了一件事，身在其中的程序员绝大多数只是负责某一块具体的业务，但我们的视野不应该只停留在自己的代码中，还要把整个调用流程梳理清楚。这里我建议你常去画画调用流程图，我个人比较喜欢画一些泳道图，可以直观地体现各个参与方。

关注上下游调用的好处就是知道流量从哪里产生，以及上游和下游的服务稳定性情况对整个链路的影响，这都是突发事件时可快速作出判断的必要条件，这里我列举几个比较重要的上下游关注点：

**产品形态。**直接面向用户的产品形态，通过直接的体验知道服务出问题时产品的报错形态。

**调用策略。**上游调用服务时的策略，是否有缓存，超时时间是多少，用户流量传递的比例。

**网络环境。**上游业务的的网络部署结构，是否跟自己的服务有跨机房、跨运营商的网络调用。

**下游服务。**下游业务的响应时间、最大抗压能力等情况，同时需要了解网络部署结构，是否有跨机房、跨运营商网络调用的情况。

2. **避免浅尝辄止**

我通过简历或者项目接触过不少同学，他们都做过高并发应用，大多数同学都能描述一些缓存系统的设计思路，整体架构大同小异，但当问到比如缓存系统占据了多少内存，命中率剔除率是多少，是怎么去规划端口数量和容量大小时，很多候选人给的回复却是我们有DBA，他帮我们维护，提供了一个 API 只需要进行服务调用就行。

但是遇到突发热点事件时，往往会因为连接数过高、带宽跑满、命中率低等问题导致整个系统负载高性能下降，如果平时不怎么关注这些，那么出现问题时我们就很难去想到应对策略，所以要想从容地应对突发事件，我们就需要调整思路、关注细节，比如缓存系统：

**关注缓存需求大小。**使用缓存时知道如何去计算需要多少内存空间，通过存储 Value 的大小、缓存的时间、活跃用户等计算出缓存的预计大小，避免存储内容增加导致缓存效率降低。

**参与缓存结构部署。**使用新缓存时学会根据存储大小、用户请求 QPS、存储时间评估出缓存部署结构，评估出需要几个实例，每个实例多少容量。通过这种方式了解缓存实例的承载上限。

**关注缓存实例运行指标。**容量、剔除率、命中率、链接数、CMD 数都是我们日常运行需要关注的指标，提前了解运行指标，可以避免突发性服务崩溃。

另外，现在很多框架、类库都对行为做了非常好的封装，在日常编码时我们只需要关注接口名和接口参数，这就导致了很多事离开了框架就无法去解释清楚。

比如我们曾经调用第三方的接口服务报错了，实现调用的代码是使用 HttpClient 封装好的库，当想快速复现一次 HTTP 请求时，我们的同学就开始为难了，一方面他没办法在线上跑起来测试程序，另一方面他的测试环境没办法快速运行起整套代码，一个简单的 HTTP 请求 curl 命令无法快速构建出来，问题就在于他不知道 HttpClient 是如何将请求发出去的，也就不知道实际调用 API 的 HTTP 请求参数是怎样的，这样遇到突发问题时就没办法从容应对。

所以平时建议你做到以下两点：

**了解底层封装逻辑。**了解相关类库底层封装逻辑，尤其是底层有许多默认参数配置，最好都去弄懂。

**抛开代码封装的实现。**时刻准备如果没有封装好的代码如何去完成同样的事情，该怎么做，可以借助什么工具。

以上这是两个很典型的例子，我们在封装精美的框架中进行开发时，美其名曰大家只需要关注自己的代码，写好自己的业务逻辑，无需操心其他，但这给大家带来的问题就是很多事情只知其一不知其二，在突发事件来临时没有足够的信息来辅助分析问题。

### 3. 平日积累

遇到突发事件时所表现出来的从容它不是一朝一夕达成的，它来自对系统、对服务的长期跟踪，来自于了如指掌的自信。

绝大多数程序员都知道写程序应该记日志加监控，但是又有多少人会定期去看监控，在没有出现问题的时候也保持去观察监控，敏锐地察觉到监控上的一点点异常呢？所以平常的积累很重要，我们可以养成一些好的习惯：

**经常看监控。**并且学会分析趋势，2 日线、7 日线是比较常关注的点，除了通过监控发现异常外，还要思考监控发现不了什么，然后做进一步的完善。

**熟悉日志。**熟悉记录的每一份日志的作用，了解日志中记录的内容，发现不合理的日志、不完善的日志，让日志真正辅助我们定位问题。

**分析日志。**线上服务尤其是高访问量的服务，日志通常会记录很多，平日查看日志的时候就不能只看单条日志，要学会基于大量的日志去分析问题。从几万条，甚至几十万条的日志中找出规律。

有了一些好习惯的积累，遇到突发事件时我们才能有条不紊地找到着手点，不至于慌慌张张不知道从哪里入手。

### 4. 一套称手的工具

对突发事件的处理讲究的就是效率，越早定位、越早解决，影响就越小，快速解决除了前面提到的积累、习惯外，还要有一套称手的工具。

**日志分析工具。**通过分析日志来定位原因是最常用到的方法，查找某一条具体的日志是最简单的，但如果需要通过日志来分析异常流量、归纳错误就涉及到大量日志的分析了。如何在面对上万条、几十万条甚至更多日志的时候不慌，那就需要考虑日志分析方法了，常用的有 awk、sed、sort、uniq 等工具，熟练使用会事半功倍。

**抓包分析工具。**网络抓包也是一个很常用的手段，分析流量从哪里来的、请求去了哪里，这时类似 Wireshark、tcpdump 等工具就派上用场了，能帮助你快速找到源头。

**系统工具。**在 Linux 系统里，有很多关于系统层面的命令，查看系统负载、网络带宽、磁盘 IO 等等，这种对于找到单机系统瓶颈很关键。

**业务工具。**业务工具有点像测试用例，一段独立的代码能帮助你快速判断代码运行结果，不少同学一定经历过遇到一个问题时需要临时去写一个脚本的状况。这种工具一般遇到问题写一个，时间长了对于快速验证是非常有帮助的。

## 成长平台

以上就是我从个人层面出发，总结出的可以快速提升应急处理能力的法门。那么从长远角度来看，我们还可以做哪些事情助力自身进阶呢？

### 1. 认清平台局限性

有些同学一毕业就去了大厂，有经验丰富的 leader 带，有平台的大规模用户，有高并发的流量，在这种背景下会有更多机会去进行技术验证，对于个人技术成长确实有很大的帮助。但是大厂我们也经常说面试的时候造火箭，实际干的是拧螺丝的活。大厂的项目很大，人员很多，所以职能划分就很细，每个人只需要关注自己的业务就能让大厂这艘船跑的很好，但如果突破不了自己，可能真就只是干拧螺丝的活。

也有同学毕业去了小厂，然后就会面临身兼数职，练就了八般武艺，但这往往就会陷入多数技术浅尝辄止、学而不精的情况，这种时候就更需要沉下心来深入学习。

可无论身在大厂还是小厂，都不要忽视习惯的养成，关心细节的习惯、关心全局的习惯、深入了解的习惯。

### 2. 清楚架构局限性

架构不是万能的，有些架构适合做业务扩展，有些架构适合高并发流量，日常开发的时候要根据业务的特性去选择架构，同时需要分析了解每一种架构的优缺点。

举个例子，我在微博参与的其中一个业务，初期的架构可以简单地称之为同步架构，遇到热点事件的时候会出现接口性能急剧下降的问题，它的瓶颈点在于同步处理耗时太多，开发的时候就需要注意服务调用策略，比如设置超时时间、自动降级等等。突发应急的时候需要快速找出同步中出问题的点，及时摘掉。

后来流量不断创新高，这个架构就有点力不从心了，所以就改成了异步架构。新架构上线后抗住了流量新高，但流量继续涨的时候还是出了问题，因为异步处理导致很多资源重复

读写，无形中增加了资源的压力，资源就成为了系统瓶颈，这个时候开发的关注点就变成了如何复用资源、如何提高资源的抗压能力，应急突发时就变成了资源扩容、替换等。

这里我想告诉你的就是，不要迷恋架构，每一个架构都有其适用性，在平日要多了解架构的优缺点，遇到突发情况时才能迅速找出痛点、作出决策、解决问题。

## 结语

在我们的工作乃至面试中，聊的最多的可能就是算法、数据结构、设计模式了，然后也会聊一些架构设计思路，比如如何保证高并发，如何保证低处理时间，如何满足很好的扩展性等等。但我在这节课里避开了这些问题，因为在服务流量一直很稳定的情况下，大概率是不会出问题的，这个只是考验系统的架构设计。

但通常在高并发服务里最常见的或者最怕见到的就是突发热点、突增流量，像微博这种系统就是经受着一次又一次的热点冲击，那么这个时候考验的就是人，快速应对、快速处理。在这一轮又一轮的冲击中，成长的除了你的架构设计能力，还有应急处理能力，从容地应对突发问题来自于脚踏实地的积累。只有夯实了自己的基础，拥有清晰的思路去分析问题，我们才能更容易地抓住问题的重点，作出相应的决策，不慌不忙从容应对。

期待你也能在诸多的问题中找到自己的应对之法！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07｜乾坤大挪移：秒杀的削峰和限流

下一篇 08｜化骨绵掌：降级、热点和容灾处理

## 精选留言 (2)

写留言

**一个卖火柴的老男人**
2021-10-15

经验比技术更重要

展开

**nana**
2021-10-12

赞

展开